REMARKS

The application has been amended and is believed to be in condition for allowance.

Claims 1-14 remain in this application.

Claims 1 and 8 have been amended, the amendments finding support in the specification and the figures (e.g. Figure 2; page 6, lines 15-19 and 25-27). The amendments introduce no new matter.

Claims 15-20 further claim the invention and find support in the figures and the specification. The new claims introduce no new matter and find support in the specification and the figures. For example, claim 18 finds support at least at page 4, lines 15-24.

Claim 12 has been amended to form. The amendment is non-substantive and adds no new matter.

An Appendix is attached for the Examiner's convenience.

The Official Action rejected claims 1-14 under 35 U.S.C. 103(a) as being unpatentable over CATALDO ("Net Processor Startup Takes Pipelined Path to 40 Gbits/s") in view of DORST (U.S. Patent Application 2004/0098549 Al).

As to claims 1 and 8, the Official Action states that CATALDO discloses a programmable pipeline adapted to directionally transfer data packets through the pipeline from a first end of the pipeline to a second end. The Official Action states that CATALDO inherently discloses plural access points located in a spaced apart relation between the first end of the pipeline and the second end of the pipeline.

The Official Action concedes that CATALDO does not disclose at least one interface engine as recited by claims 1 and 8.

The Official Action states that DORST discloses at least one interface engine as recited in claims 1 and 8. The Official Action states that the teaching of DORST reduces the burden and overhead of processors interfacing with and controlling the memory and flexibly controls a multitude of memory circuits in a simple-to-use manner. The Official Action concludes that it would have been obvious to one of ordinary skill in the art at the time of the invention to combine the teaching of DORST with the invention of CATALDO in order to reduce the burden and overhead of processors interfacing with and controlling the memory, as well as flexibly control a multitude of memory circuits in a simple-to-use manner.

Applicants respectfully disagree.

It is respectfully submitted that neither DORST nor CATALDO, individually or in combination, teach or suggest a processor having at least one interface engine adapted to be connected to at least one external device located externally of the processor and connected to each of a plurality of access points between a first end of the pipeline and a second end of the pipeline, as required by claim 1.

On the contrary, DORST teaches a memory controller 1005 connected to at least one memory module 1015A,1015B,1015N (Figure 3) and a processor 1010 (Figure 1; paragraph [0010]). In particular, a memory controller resides within a processor (paragraph [0031], lines 3-4). Depending on the application and desired performance, DORST may incorporate more than one processor 1010 and/or more than one memory controller 1005, and furthermore, several processors 1010 may share a memory controller 1005, or vice-versa (paragraph [0031, lines 5-15]).

DORST may therefore disclose an embodiment wherein a memory controller is connected to each of a plurality of processors. DORST does not teach or suggest a programmable pipeline, as conceded by the Official Action. Accordingly, neither the processors nor the connections to the processors teach or suggest plural access points of a programmable pipeline.

CATALDO teaches a programmable pipelined network processor (CATALDO paragraph 1, lines 2-3). The pipeline is divided into plural stages wherein each stage acts as a mini-processor (CATALDO paragraph 5, lines 3-7). Each stage in the pipeline performs a classification and action on incoming networking packets such that packets may be disseminated and processed at "wire speed," that is, responsive to packets as they come in (CATALDO paragraphs 1, lines 3-5).

It is respectfully submitted there is no teaching or suggestion of plural access points within CATALDO's pipeline.

The Official Action states that CATALDO inherently discloses plural access points located in a spaced apart relation between the first end of the pipeline and the second end of the pipeline, asserting that the "access points could simply be polls to each stage or the like," (Official Action, page 2, paragraph 5, lines 6-9).

Applicants respectfully find no support for this in the reference. It is further submitted that, at best, the access points alleged by the Official Action do not teach or suggest access points adapted to be connected to at least one interface engine and transmit a request to the interface engine upon arrival of a data packet, as required by claims 1 and 8, and further to receive a response from the interface engine as required by dependent claims 5 and 12.

In contrast, CATALDO discloses a pipeline adapted to handle packets "as they come in" (paragraph 3, lines 2-3), performing classification and action operations at every stage to get wire-speed performance (paragraph 5, lines 1-3). There is no teaching or suggestion of an access point, <u>between</u> the first end and the second end, sending a request to an interface engine as required by claim 1, and then receiving a response as required by claim 5 (also see claim 17). On the contrary, an access point as recited by the claims would cause the pipeline of CATALDO to <u>wait</u>, halting the pipeline while the interface engine processes the request, thereby forfeiting the inventive feature offered by CATALDO of processing network packets <u>at wire speed</u>.

Accordingly, it is respectfully submitted that neither CATALDO nor DORST, individually or in combination, teach or suggest the pipeline and access points as recited by claims 1 and 8.

It is further respectfully submitted that neither CATALDO nor DORST, individually or in combination, teach or suggest an interface engine connected to <u>each</u> of plural access points of a programmable pipeline <u>and</u> connected to an external interface, as required by claims 1 and 8.

In contrast, DORST teaches a memory controller 1005 connected to a plurality of memory modules 1015A,1015B,1015N (Figure 3) and a processor 1010 (Figure 1; paragraph [0010]). DORST further discloses that system 1000 may have more than one processor 1010 and/or more than one memory controller 1005 (paragraph [0031]). DORST provides no teaching or suggestion of a programmable pipeline having plural access point, each access point connected to one or more memory controllers, and CATALDO, as stated above provides no teaching or suggestion of plural access points between the first and second ends of the pipeline configured to interface with one or more memory controllers.

Accordingly, it is respectfully submitted that neither CATALDO nor DORST, individually or in combination, teach or suggest an interface engine as recited in claims 1 and 8, and further, one of skill would not be motivated to combine the teachings of these references as there is no teaching or suggestion in CATALDO that its sequential pipelined processor would benefit or interface with DORST's memory controller providing random access between processors and memory controllers.

It is further submitted that neither CATALDO nor DORST, individually or in combination, teach or suggest an interface engine comprising an arbiter configured to allow forwarding of requests from the plurality of access points in a fair manner between each of the plurality of access points, as required in claims 1 and 8 as amended (see specification page 6, lines 15-19 and 25-27; also, please see Luciano Lenzini, "Eligibility-Based Round Robin for Fair and Efficient Packet Scheduling in Wormhole Switching Networks", appended to this response). As stated above, CATALDO does not teach or suggest access points between the first end and the second end of the pipeline, DORST does not teach or suggest a pipeline, and there is no teaching or suggestion in either reference of an arbiter that might interface with intermediate access points of a pipeline to forward packets from them in a fair and cyclic manner.

For all the foregoing, it is respectfully submitted neither CATALDO nor DORST, individually or in combination, teach or suggest all the features recited in independent claims 1 and 8. Therefore, it is respectfully submitted that independent claims 1 and 8, and all claims dependent therefrom, are patentable.

Reconsideration and withdrawal of the rejection are respectfully requested.

New claims 15-19 are respectfully submitted as patentable for at least the reasons stated above. Allowance of the new and previously presented claims is earnestly requested.

From the foregoing, it will be apparent that applicants have fully responded to the October 16, 2007 Official Action and that the claims as presented are patentable. In view of this, applicants respectfully request reconsideration of the claims, as presented, and their early passage to issue.

In order to expedite the prosecution of this case, it is requested that the Examiner telephone the attorney for applicant at the number set forth below if the Examiner is of the opinion that further discussion of this case would be helpful.

The Commissioner is hereby authorized in this, concurrent, and future replies, to charge payment or credit any overpayment to Deposit Account No. 25-0120 for any additional fees required under 37 C.F.R. § 1.16 or under 37 C.F.R. § 1.17.

Respectfully submitted,

YOUNG & THOMPSON

[illegible]

Roland E. Long, Jr., Reg. No. 41,949
745 South 23rd Street
Arlington, VA 22202
Telephone (703) 521-2297
Telefax (703) 685-0573
(703) 979-4709

REL/lk

APPENDIX:

The Appendix includes the following item:

- article by LENZINI et al. entitled "Eligiblity-Based Round Robin for Fair and Efficient Packet Scheduling in Wormhole Switching Networks"

Eligibility-Based Round Robin for Fair and Efficient Packet Scheduling in Wormhole Switching Networks

Luciano Lenzini, *Member, IEEE*, Enzo Mingozzi, and Giovanni Stea

Abstract—Interconnection networks of parallel systems are used for servicing traffic generated by different applications, often belonging to different users. When multiple users contend for channel bandwidth, fairness in bandwidth sharing becomes a key requirement. In fact, enforcing a fair sharing of channel bandwidth improves flow isolation, thus preventing misbehaving flows from affecting the performance of other flows. This paper presents a novel packet scheduling algorithm, called Eligibility-Based Round Robin (EBRR), devised to provide fair queueing in interconnection networks. In fact, EBRR meets the constraints imposed by wormhole switching, which is the most popular switching technique in interconnection networks of parallel systems. It can also be applied to packet switching Wide Area Networks (WANs), such as IP and ATM. We show that EBRR has O(1) complexity and better delay and fairness properties than existing algorithms of comparable complexity. In this paper, we also investigate the means for assessing the fairness of a scheduler: We show that using the Relative Fairness Bound as a fairness measure may lead to erroneous results. We then propose an alternative measure, called the Generalized Relative Fairness Bound, that allows fairness to be assessed more precisely.

Index Terms—Packet scheduling, fairness, wormhole switching networks, quality of service.

1 Introduction

Interconnection networks of parallel systems are used for servicing traffic generated by different applications, often belonging to different users [17], [19], [20]. When multiple traffic flows contend for channel bandwidth, the scheduling algorithm regulating the access to that channel plays a key role in ensuring that each flow obtains the required quality of service. Fairness is a highly desirable property for a scheduling algorithm. The notion of *fairness* is somewhat intuitive: A scheduling algorithm is *fair* if it shares bandwidth among competing flows according to the respective rate requirements, so that the rate ratio matches the rate requirement ratio. A fair scheduling algorithm prevents a misbehaving flow from obtaining more than its share of the bandwidth, thus possibly starving its competitors. Fairness also improves flow isolation, which is important in order to obtain predictable performance. Another important attribute of a scheduling algorithm is its ability to guarantee an upper bound on the delay experienced by packets. A tighter bound on the delay favors lower communication latencies, thus reducing the overall computation time in parallel processing applications. Finally, in order for a scheduling algorithm to be applicable when scheduling decisions involve many flows and have to be taken quickly (due to the high bandwidth of the output channels), it is desirable that the number of operations involved in a scheduling step in the worst-case be *constant* with respect to the number of flows. If this is the case, an algorithm is said to have $O(1)$ *worst-case per-packet complexity* (hereafter, *complexity*). Note that algorithms which require a low number of operations per packet transmission *on average* might still require unsuitably heavy computations in the worst-case. All these attributes are simultaneously required in emerging fields of applications of parallel and distributed systems, such as virtual reality, multimedia and Web servers, and collaborative interactions [20].

Whereas WANs are customarily packet switching (or *store and forward*), *wormhole switching* and its variations are more often employed in interconnection networks of parallel systems. In wormhole switching networks, a packet is not entirely buffered at a single node when it cannot be forwarded because the output channel is occupied. Instead, portions (*flits*) of the same packet are stored at consecutive nodes along the packet route. When the packet is forwarded, each flit advances along the route accordingly. Note that it may not be possible to multiplex flits from different packets on the same channel. This has two important consequences. First of all, when the head flit of a packet is blocked because the channel is busy, the block propagates backward to the upstream channels on which the remaining flits were being transmitted. Thus, a channel might remain unutilized due to a packet being blocked at a downstream node. This also means that knowing the *length* of a packet does not imply knowing how long it will hold the channel (hereafter, its *service time*) since the latter might include blocking times. Second, the decision about whether to schedule a packet at a node has to be taken based on the information contained in the *header flit* of the packet. In some wormhole switching networks (e.g., Myrinet [17]), the header flit does not contain a packet length field. Therefore, in order to be suitable for wormhole

• *The authors are with the Dipartimento di Ingegneria dell'Informazione—University of Pisa, Via Diotisalvi 2, 56122 Pisa, Italy.*
E-mail: {l.lenzini, e.mingozzi, g.stea}@iet.unipi.it.

Manuscript received 1 Aug. 2002; revised 3 Apr. 2003; accepted 17 Aug. 2003.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number 117053.

switching networks, a scheduling algorithm should be able to decide whether to start the transmission of a packet without knowing either its length or its actual service time in advance.

The fact that in wormhole switching networks the length of a packet and its service time are not directly related to each other, also requires some comments on the definition of fairness. Whenever packets are transmitted on a channel which forwards traffic at a *constant* rate (as is often the case in packet switching WANs), fairness can be assessed by comparing either the time interval during which competing flows hold the channel or the *service* (i.e., the number of transmitted bytes or flits) that they receive in a given time interval. In wormhole switching networks, on the other hand, the actual rate at which a channel transmits flits varies over time, and is either zero (when a packet is blocked) or the full channel bandwidth. When dealing with such *variable rate servers* [6], fairness has to be assessed by comparing the *service* received by flows, rather than how long they hold a channel [6], [7]. Consider, for example, the case in which two identical flows i and j send packets along different routes, sharing a single output channel along their respective routes. Suppose that the two flows experience a different blocking probability, e.g., $P(i) > P(j)$, due to one route being more congested than the other. In that case, flow i needs to occupy the shared channel for a longer time than flow j in order to receive the same service. Therefore, a scheduler that tries to equalize the length of time in which the two flows hold the channel would be unfair. In order to be *fair*, a scheduling algorithm devised for a wormhole switching network should therefore try to equalize the *service* received by flows. Consequently, hereafter, we will say that a flow is receiving service if its packets are actually being forwarded on the output channel.

The contribution of this paper is twofold. First, we define and analyze a novel scheduling algorithm, called Eligibility-Based Round Robin (EBRR). Although EBRR can be used in a variety of contexts, e.g., for scheduling in packet switching WANs, it also meets the specific requirements of wormhole switching technology. We prove that EBRR has $O(1)$ complexity, and is thus suitable for operating at high speeds and with a large number of flows, and we analyze its delay and fairness properties. In order to do so, we need suitable means for assessing the fairness of a scheduler. A commonly used measure of a scheduler's fairness is its *Relative Fairness Bound* (RFB), first proposed in [5] and widely adopted in the literature (see, for example, [2], [8], [10], [14]). The latter is defined as the maximum difference in the normalized[1] service received by any two flows over any time interval in which they are both backlogged. Second, we prove that assessing a scheduler's fairness on the basis of its RFB may mean that a scheduler is judged as being fair when in fact it is not. We then propose an alternative fairness measure, which we call *Generalized Relative Fairness Bound* (GRFB), that allows us to assess the fairness of a scheduler more precisely than the RFB.

The rest of the paper is organized as follows: In Section 2, we formulate the scheduling problem and discuss the related work. In Section 3, we describe the EBRR scheduling algorithm. We analyze EBRR in Section 4, where we also introduce the GRFB. We report simulation results in Section 5 and, in Section 6, we draw conclusions and highlight areas for future work.

1. For instance, divided by the rate required by the flow.

2 Related Work

A scheduling problem exists whenever there is a shared resource that needs to be arbitrated among multiple requesting entities. However, even though the problem is general, its solution may be very different depending on the specific context it is tailored on. In this section, we first discuss whether existing scheduling algorithms—which may have been devised for different contexts—can be employed in interconnection networks. We then discuss solutions explicitly devised for wormhole switching networks.

Over the last 10 years, the problem of fair packet scheduling in WANs has received much attention (see, for example, [1], [2], [3], [4], [5], [6], [7], [8], [9], [10], [11], [12], and the works referred to in [1]). The ideal Generalized Processor Sharing (GPS) [4], which achieves perfect fairness in a fluid-flow system by servicing all backlogged flows simultaneously according to their weights, has been proposed as a reference model for devising *packet fair queueing* schedulers. A wealth of packet scheduling algorithms approximating GPS have then been proposed (e.g., [4], [5], [6], [8]), each with a different trade off in terms of accuracy in GPS approximation and implementation complexity.

The existing packet fair queueing algorithms can be classified as being either *sorted-priority* or *frame-based* [2]. Sorted-priority algorithms associate a timestamp with each queued packet and transmit packets by increasing timestamp order. On the other hand, frame-based algorithms divide time into frames and select which packet to transmit on a per-frame basis. Within the frame-based class, *round-robin* algorithms service the various flows cyclically, and within a round each flow is serviced for up to a given *quantum*, so that the frame length can vary up to a given maximum. Sorted-priority algorithms generally exhibit better fairness properties than round-robin ones, but have a higher complexity due to the calculation of the timestamps and to the sorting process [2]. Specifically, the task of sorting packets from N different flows has an intrinsic complexity of $O(\log N)$. In interconnection networks of parallel systems, such complexity would not be suitable. Moreover, the timestamp computation requires knowledge of the length of a packet which, as already observed, is not always possible in wormhole switching networks. On the other hand, round-robin algorithms, such as Deficit Round Robin (DRR) [8] and Surplus Round Robin (SRR) [9], can exhibit $O(1)$ complexity under specific conditions. In both algorithms, backlogged flows are stored in a FIFO list (called the *active list*). Cyclically, flows are dequeued from the head of the active list, serviced, and—if still backlogged—enqueued back at its tail. In DRR, each flow maintains a *deficit counter*, which is increased by the flow's quantum on each round (i.e., whenever the flow is dequeued from the active list) and decreased by the length of each transmitted packet. Whenever a flow is serviced, the transmission of a packet is allowed as long as it does not make the deficit fall below zero. Therefore, if the head packet length is larger than the remaining deficit, the flow's turn ends and the remaining deficit is carried over onto the next round as a credit. SRR works the other way around: flows have a *surplus counter*, managed exactly like the deficit counter in DRR; in SRR, a flow is allowed to transmit packets as long as its surplus is positive. When the surplus becomes nonpositive, the flow's turn ends and the remaining surplus is carried over onto the next round as a debit. DRR and SRR have $O(1)$ complexity *provided that* each flow is allocated a quantum larger than its maximum packet length [8]. Working with large quanta implies allowing flows to transmit large bursts of packets, which, in turn, affects fairness. DRR and SRR have the same RFB [3]. However, DRR is not applicable to wormhole switching networks since it requires knowledge of the length of a packet before deciding whether to transmit it, whereas in SRR, such knowledge is not required.

Preorder Deficit Round Robin (PDRR) [21] aims at emulating the Packet Generalized Processor Sharing (PGPS) [4] by coupling a DRR module with a priority module that manages Z FIFO priority queues. Each packet which should be transmitted in the current DRR round is instead sent to one of such priority queues, selected according to the expected packet leaving time under PGPS. A static priority scheduler regulates the actual transmission of the packets by dequeueing them from the priority queues. Therefore, the set of packets that would leave in a DRR round is "preordered" before their transmission takes place. The higher the number of priority queues Z is, the closer PGPS is approximated and the more PDRR improves on DRR as for fairness and delay bounds. However, this comes at the expenses of an increase in the per-packet complexity. In fact, in order to find, on every scheduling step, the nonempty priority queue with the highest priority, a min heap is used to sort references to priority queues. In [4], authors claim that PDRR retains the DRR $O(1)$ complexity, the worst-case overhead per packet transmission being the $O(\log Z)$ cost of insertion in the min heap. However, according to the pseudocode, at the beginning of a round, the min heap structure is empty and it has to be filled up by dequeueing packets from *all* backlogged flows and sending each of them to the relevant priority queue. This process has to be completed *before* the transmission of a packet is started, otherwise, it can be easily proven that the PDRR reordering would be thwarted. Looping through all backlogged flows requires $O(N)$ iterations, Z of which might trigger a min heap insertion. Thus, transmitting the first packet in a round requires $O(N + Z \log Z)$ operations, which is even larger an order of magnitude than that of many sorted-priority schedulers. Moreover, PDRR inherits DRR's inapplicability to wormhole switching networks.

Elastic Round Robin (ERR) [14] has recently been proposed as a fair and efficient scheduling algorithm explicitly devised for wormhole switching networks. In ERR, flows are not serviced for a fixed quantum on each round; instead, a backlogged flow computes its *allowance*, i.e., the minimum number of flits that it will be allowed to transmit (if its backlog is large enough), on a per-round basis. A flow transmits packets as long as its allowance is positive, and the flow's *surplus counter* SC_i records the possible overrun due to the transmission of the last packet, similarly to SRR. At the end of a round, the maximum value achieved by a surplus counter during the ongoing round is copied into the global variable *PreviousMaxSC*. The allowance for flow i at the beginning of a round is then computed as: $A_i = 1 + PreviousMaxSC - SC_i$. Therefore, a flow which accumulates a large surplus during a round is penalized in the subsequent one. Whenever a flow

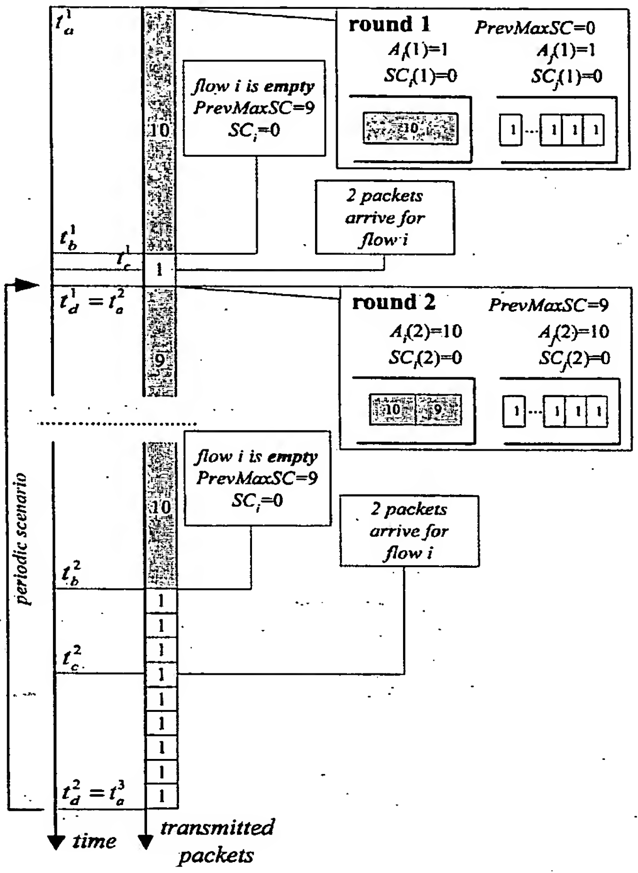

Fig. 1. Example of ERR unfairness.

becomes idle, its surplus counter is reset to zero. The compliance of ERR with respect to the wormhole switching constraints is warranted by the fact that the allowance of a flow is decreased *after* a packet is transmitted. In [14], the authors prove that ERR has $O(1)$ complexity, delay bounds which are tighter than those of DRR and SRR, and a *finite* RFB. However, we show that ERR can actually exhibit an *unfair* scheduling behavior, allowing a flow to steal bandwidth from its competitors. Suppose that two identical flows i and j are scheduled with ERR. One would expect from a fair algorithm that, as long as j is continuously backlogged, it is serviced at a rate no lower than half the channel speed, regardless of the behavior of flow i. We show that this is not the case: we describe a simple scenario—a graphic representation of which is reported in Fig. 1—in which flow i permanently overdraws bandwidth, transmitting its packets at a rate which is approximately double that of flow j.

In fact, suppose flows i and j become active at the beginning of round 1, at time t_a^1 (superscript denotes the round number, while subscript denotes the order of occurrence of an event within a round). Their allowance for round 1 is then $A_i(1) = A_j(1) = 1$. Suppose that flow i has a 10-flit packet (and nothing else) in its packet queue, whereas flow j has a large supply of 1-flit packets queued. Flow i transmits its 10-flit packet, thus becoming idle, at time $t_b^1 > t_a^1$. SC_i is then reset to zero at time t_b^1, which implies forgetting about its nine flits excess service. Flow j then transmits one packet, thus achieving $SC_j = 0$ at time $t_d^1 > t_b^1$. Suppose now that, at time $t_c^1 \in (t_b^1, t_d^1)$, flow i becomes backlogged again, with a 9-flit packet followed by a 10-flit packet. It will then be serviced during round 2, starting at time $t_d^1 = t_a^2$.

At time t_a^2, we have $PreviousMaxSC = 9$, and the allowance of the two flows is: $A_i(2) = 1 + 9 - 0 = 10$, $A_j(2) = 1 + 9 - 0 = 10$. Thus, flow i transmits *both* its packets, for an overall 19 flits length and empties its packet queue, finishing at time t_b^2; SC_i is then reset to zero. Then, flow j transmits 10 packets, for an overall 10 flits length, finishing at time t_d^2. Suppose again that a 9-flit and a 10-flit packet arrive at flow i at time $t_c^2 \in (t_b^2, t_d^2)$, i.e., while flow j is being serviced. It is easy to see that the same scenario described for round 2 may be repeated for an arbitrary number of consecutive rounds, thus allowing flow i to transmit packets at a rate which is almost *double* that of flow j.

The unfairness of ERR is due to the fact that surplus counters are *reset* as soon as flows become idle. This implies "forgetting" about any excess service received by idling flows, which are allowed to compete for the channel bandwidth again as early as in the subsequent round. In fairness to the authors, we observe that the problem analyzed in the example might be removed by avoiding resetting surplus counters of idling flows. Moreover, the $PreviousMaxSC$ variable should be set to the maximum between its value *and* that of the surplus counter of any flow which becomes backlogged during the round just elapsed. However, we also observe that this would increase the allowance for each flow, which, in turn, would lead to burstier transmissions and higher average delays. Moreover, this would penalize a flow which becomes backlogged, thus possibly increasing its queueing delay.

It may be worth noting that the above-mentioned problem might also affect SRR since, in this algorithm, flows are allowed to receive excess service. However, no pseudocode or statement explaining how surplus counters are managed when flows become idle is reported in [9] and, therefore, we cannot draw conclusions. Also note that SRR has a finite RFB no matter whether surplus counters are reset or not when a flow becomes idle, since the RFB is computed in time intervals in which flows are continuously backlogged. The above considerations also prove that assessing a scheduler's fairness on the basis of its RFB can lead to erroneous conclusions. In fact, it proves that having a finite RFB is not a sufficient condition for avoiding an unfair scheduling behavior.

3 Eligibility-Based Round Robin

We now introduce the EBRR scheduling algorithm. In order to make the formulation of the problem clearer and more general, we present the following abstraction of the scheduling problem. We assume that N flows[2] contend for transmission on a channel whose capacity is C. Each flow i requires a minimum rate ρ_i, such that $\rho_1 + \ldots + \rho_N \leq C$, and has a dedicated FIFO packet queue. We assume that packet lengths are expressed as numbers of flits. A flow is *active* (or *backlogged*) if at least one packet is buffered in its own queue. A scheduling step consists of selecting the next active flow whose head-of-line packet has to be dequeued and transmitted next. In a wormhole switching network, packets are not physically stored in a packet queue at each node (as is the case in packet switching networks) since they occupy flit buffers across consecutive nodes along the packet route. Therefore, in that case, *enqueueing* a packet means making the scheduler aware that the flow the packet belongs to is active, and *dequeueing* it means deciding that the packet will be the next to be transmitted. Moreover, in a wormhole switching network, we consider that a packet has *arrived* at a node (and can therefore be enqueued) if its header flit has been completely received, as opposed to the case of packet switching networks in which a packet is usually considered as having arrived when its *last* bit has been received.

According to EBRR, each flow i is assigned a quantum ϕ_i, expressed as a number of flits, and a *credit counter* C_i. Quanta assigned to flows are proportional to their respective rate requirements, i.e., $\phi_i/\phi_j = \rho_i/\rho_j$, $\forall i, j$. Ideally, a flow should receive an amount of service equal to ϕ_i flits per round in order to utilize the channel with at least the required capacity ρ_i. However, since packets are of variable length and flits from different flows cannot be multiplexed on the same channel, it might not be possible to match the actual service received by flow i to the quantum value ϕ_i exactly in each round. For this reason, a credit counter C_i is used, which keeps an account of the service that flow i is still entitled to receive in the current round. The first time that flow i is selected for service in a round,[3] C_i is incremented by the quantum value ϕ_i. Whenever flow i is serviced, the head-of-line packet from flow i is transmitted and, *after that*, C_i is decremented by the packet length. According to EBRR, each flow i is allowed to overdraw its account, i.e., C_i can take negative values. However, as soon as C_i becomes negative or null, flow i is not eligible to receive any further service in the current round.

So far, EBRR and SRR manage credit counters in the same manner. The new features introduced by EBRR are specifically related to the selection of the next flow to be serviced. As mentioned above, in SRR, in order to ensure an O(1) per packet complexity, the quantum value must be no smaller than the largest packet that may potentially arrive at the queue. The reason is basically that a single list of active flows is maintained. Whenever the flow at the head of the list is selected for service, its credit counter is incremented by the quantum value and, *if the counter becomes positive*, the head-of-line packet is transmitted and the credit counter is updated accordingly. The flow is then inserted back at the end of the queue. As a consequence, it may happen that for several consecutive times (even more than N in the worst-case), the flow selected for service is inserted back without any packet transmission until the credit counter becomes positive. However, if flows, although active, could be removed from the list as soon as they become ineligible for transmission, the above situation would be avoided

2. The way in which packets are *classified* as pertaining to a given flow depends on the network architecture. As this issue is clearly independent of the scheduling policy, it is outside the scope of this paper.

3. As will be explained later on, a flow may receive service more than once in a round according to EBRR.

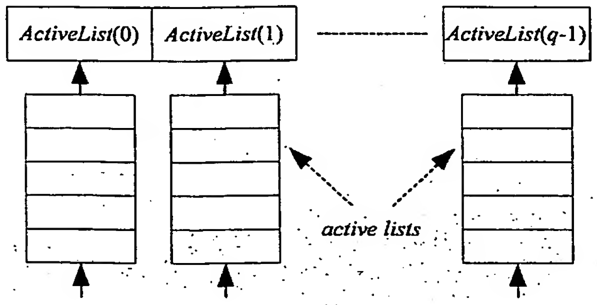

Fig. 2. Array of active lists.

even though the quantum value might be smaller than the length of a packet. To this end, EBRR maintains an array of $q > 1$ lists of active flows, as shown in Fig. 2. The kth list in the array is referred to as *ActiveList*(k). The lists refer to q consecutive different rounds, which we assume are numbered by consecutive positive integers.[4] A *round counter* RC is used to represent the number of the current round. According to EBRR, *ActiveList*($|RC|_q$), hereafter the *current* list, includes flows that are still eligible in the current round, while *ActiveList*($|RC+k|_q$), $0 < k \leq q-1$, includes flows that are no longer eligible at round RC, but will become eligible at round $RC+k$.

According to the procedure described above, the scheduler selects for service only flows that are eligible in the current round, i.e., those flows included in the current list. Let us assume that flow i is the last one that was selected for service and, after updating the counter C_i, the flow is no longer eligible in the current round, i.e., $C_i \leq 0$. But, then, the next round EC_i at which the flow will be eligible again can be soon computed as follows:

$$EC_i \leftarrow RC + 1 + \left\lfloor \frac{-C_i}{\phi_i} \right\rfloor \tag{1}$$

since the credit counter increases by one quantum per round. As a consequence, flow i is removed from the current list and inserted into *ActiveList*($|EC_i|_q$). We will refer to EC_i as the *eligibility counter* of flow i. Furthermore, the value of the credit counter can be immediately updated to the one expected at the time the flow becomes eligible again, according to the following formula:

$$C_i \leftarrow C_i + \left(1 + \left\lfloor \frac{-C_i}{\phi_i} \right\rfloor\right) \cdot \phi_i. \tag{2}$$

It clearly follows that a new round starts whenever the current active list is empty and, hence, the next nonempty list is considered for service. At this point, the round counter RC can be updated accordingly. We will address in the next section the problem of determining under what conditions, by maintaining an array of active lists, EBRR operates correctly. We will also prove that, under those conditions, its complexity is O(1).

As mentioned above, our purpose is to improve fairness as well. We are now ready to complete the algorithm definition by introducing those features that are particularly relevant for fairness. First of all, whenever an eligible flow is selected for service, instead of transmitting multiple packets back-to-back from that flow until either it is no longer eligible or the flow queue is emptied out, the flow is allowed to transmit *one* packet only. After that, if the flow is still eligible, it is inserted back into the current list so that every other eligible flow (if any) is allowed to transmit a packet before the flow can transmit again. It is clear that, since ineligible flows are removed from the current list, this implementation does not affect which packets are transmitted within a round, but just the order with which they are actually transmitted.

Second, we want to avoid the same flaw that produces unfairness in ERR (described in Section 2). To this end, whenever a flow i becomes active after a packet arrival at an empty queue, the list into which the flow is inserted depends on how much surplus service, if any, the flow received the last time that it was active and completed the transmission of packets in its queue. Specifically, consider the round at which flow i transmitted the last packet in the queue before becoming idle. Now, either the flow was still eligible after transmitting the packet, or it was no longer eligible. In the former case, as soon as the flow is active again, it is allowed to immediately join the current list since it did not overdraw its account in the last transmission. On the other hand, in the latter case, we want to account for the service that the flow received although it was not entitled to so that the flow is prevented from taking advantage of such situations. To this end, the next round EC_i at which the flow would have been eligible is computed according to (1), as if the flow were still active. When the flow becomes active again, we do not want it to be serviced *earlier* than EC_i. As a consequence, if RC is greater than EC_i, the flow is inserted into the current list and its credit is reset to the quantum value. Otherwise, it is inserted into *ActiveList*($|EC_i|_q$), and the credit is recomputed as if the flow had never become idle. The pseudocode for EBRR is reported in Fig. 3.

The *FlowInit* function is invoked whenever a new flow is activated. It sets its credit to the quantum and its eligibility counter to the number of the current round, that is, the activating flow will be eligible as soon as it becomes backlogged. The *PacketArrival* function is invoked whenever a new packet arrives at the scheduler. If the flow was previously idle, it is now backlogged and has to be inserted into one of the active lists. As explained above, if the flow has been idle for longer than it should have been in order to become eligible again, i.e., $RC > EC_i$, its credit is set to the full quantum and its eligibility counter is reset to the current round. In any case, the flow is inserted into *ActiveList*($|EC_i|_q$). The incoming packet is then enqueued in the flow's packet queue.

Finally, the *EBRRScheduler* function is the core of the algorithm. It is invoked whenever there are packets to be scheduled. As long as the current list is not empty, the head-of-line flow i is selected for service. The head packet of flow i is dequeued and transmitted, and its length (after the packet transmission) is subtracted from flow i's credit. If flow i is no longer eligible in the current round, i.e., $C_i \leq 0$, the next round at which it will be eligible again is computed according to (1) and stored in EC_i, C_i is set to the credit expected at round EC_i, and the flow, if already active, is inserted into *ActiveList*($|EC_i|_q$). When the current list is

4. Note that, although at a first sight the EBRR active list mechanism might look similar to that of PDRR priority queues, we observe that they are, in fact, different, as the former contain references to backlogged flows, whereas the latter contain packets.

```
FlowInit (flow $i$) {
    $C_i = \phi_i$;
    $EC_i = RC$;
}

PacketArrival (packet $a$, flow $i$) {
    if (Idle($i$)) {
        if ($RC > EC_i$) {
            $C_i = \phi_i$;
            $EC_i = RC$;
        }
        EnqueueFlow($i$, ActiveList($EC_i$ mod $q$));
    }
    EnqueuePacket($a$, PacketQueue($i$));
}

EBRRScheduler() {
    while (SystemBusy) {
        while (NotEmpty(ActiveList($RC$ mod $q$)) {
            $i$ = DequeueFlow(ActiveList($RC$ mod $q$));
            $a$ = DequeueHeadPacket($i$);
            TransmitPacket($a$);
            $C_i = C_i -$ Length($a$);
            if ($C_i \le 0$) {
                $EC_i = RC + 1 +$ floor($-C_i / \phi_i$);
                $C_i = C_i + (1 +$ floor($-C_i / \phi_i$)) $\cdot \phi_i$;
            }
            if (Active($i$))
                EnqueueFlow($i$, ActiveList($EC_i$ mod $q$));
        }
        $RC$ = NewRound();
    }
}
```

Fig. 3. Pseudocode for EBRR.

empty, a new round is started. The *NewRound* function is intentionally left unspecified at this time, as the high-level description of EBRR does not depend on its implementation; in Section 4.1, we discuss its possible implementations.

4 Analysis

In this section, we analyze the EBRR scheduling algorithm. First, we present the assumptions under which EBRR is analyzed. We then investigate its complexity, fairness, and delay bounds. Let us denote with F (*frame length*) the sum of the quanta, i.e.,

$$F = \sum_{j=1}^{N} \phi_j. \tag{3}$$

Therefore, the target rate at which a flow should be serviced is given by:

$$\rho_i = C \cdot \frac{\phi_i}{F}. \tag{4}$$

We assume that each flow i, $i = 1 \dots N$, sends packets whose length is between 1 and L_i flits. In EBRR, the maximum packet length is related to the quantum selection and the number of active lists employed. Consider flow i, and suppose that it becomes backlogged or it is still backlogged after transmitting a packet; in both cases, it is enqueued into $ActiveList(|k|_q)$, where $k \le RC + 1 + \lfloor -C_i/\phi_i \rfloor$. This is because we want the flow to become eligible after exactly $1 + \lfloor -C_i/\phi_i \rfloor$ rounds have elapsed. Therefore, in order for EBRR to operate correctly, we need to ensure that $ActiveList(|k|_q)$ is visited after $1 + \lfloor -C_i/\phi_i \rfloor$ rounds, i.e., we need to enforce the following inequality:

$$1 + \lfloor -C_i/\phi_i \rfloor < q. \tag{5}$$

Let us prove a straightforward result.

Lemma 1. *The following inequalities always hold in EBRR:*

$$-(L_i - 1) \le C_i \le \phi_i. \tag{6}$$

Proof. Let us prove the two inequalities separately:

1. $-(L_i - 1) \le C_i$. When flow i is about to transmit a packet, its credit is positive (otherwise, the flow could not be located in the current list). Since we consider integer quanta and packet lengths, flow i's credit is at least one when a packet transmission begins. Since the packet length for flow i is at most L_i, we straightforwardly obtain the thesis.
2. $C_i \le \phi_i$. A flow's credit may increase when the flow becomes backlogged and after it has transmitted a packet. In both cases, we might have either $C_i \leftarrow \phi_i$ (and, therefore, the inequality holds) or $C_i \leftarrow C_i + (1 + \lfloor -C_i/\phi_i \rfloor) \cdot \phi_i$, which yields $C_i \le \phi_i$ for any $\phi_i > 0$.

This completes our proof. □

On the basis of Lemma 1, we can rewrite inequality (5) as follows:

$$q \ge 2 + \max_{i=1..N} \left\lfloor \frac{L_i - 1}{\phi_i} \right\rfloor. \tag{7}$$

Inequality (7) implies $q \ge 2$, which can be intuitively explained by considering that at least two active lists are needed in EBRR in order to differentiate between eligible and noneligible flows. As a consequence of (7), a flow's quantum need not necessarily be greater than its maximum length packet. In fact, if we select $q > 2$, we can allow a flow's quantum to be smaller than its maximum length packet by $q - 1$ times. In the rest of the paper, we assume that q is given and quanta are selected so that (7) holds.

4.1 Complexity

It can be straightforwardly proven that EBRR has $O(1)$ complexity. In fact, the active list queue is a collection of FIFO lists and, therefore, enqueueing and dequeueing flows in a specific list are $O(1)$ operations once it is known which list is involved. Suppose that EBRR is servicing a flow during round RC (so that the involved list is $ActiveList(|RC|_q)$). All flows queued in the current list are backlogged and eligible and, therefore, at least one packet is transmitted whenever a flow is dequeued. On the other hand, locating the correct active list in which to enqueue a flow, either when it becomes backlogged or if it is still backlogged after being serviced, is an $O(1)$ operation as well. Thus, the complexity of EBRR is constant with respect to the number of scheduled flows. When the flows in the current list have all been serviced, a new round starts. In that case, the scheduling step requires the next nonempty list from which to dequeue a backlogged flow to be located (which, in the pseudocode of Fig. 3, is the action performed by the *NewRound* function). There are several ways to efficiently accomplish this task depending on the context in which EBRR is to be applied. If EBRR is to be implemented in hardware (as it normally is in interconnection networks), it is possible to use a flag bit for each active list, so that this bit is set whenever the list is not empty, and a priority encoder can be used to implement the *NewRound* function. If EBRR is to be implemented in software (as it frequently is in IP or ATM networks), locating the next nonempty list might be a time-critical task, as it requires a number of operations which depend on the number of active lists q. Though we do not show it here, additional data structures can be employed to accomplish this task at a cost of $O(\log q)$, or even $O(\log\log q)$operations,[5] thus allowing one to efficiently operate with a large number of active lists. Some further considerations on this topic may be found in [10].

4.2 Fairness

In this section, we discuss the limits of the Relative Fairness Bound and introduce the Generalized Relative Fairness Bound. We then assess the fairness of EBRR by computing its GRFB. In the introduction, an informal definition of RFB was provided. Let us thus start by giving its formal definition.

Definition 1. *The relative fairness bound $RFB_{i,j}$ between any two flows i and j is:*

$$RFB_{i,j} = \max_{t_2>t_1}\left|\frac{W_i(t_1,t_2)}{\rho_i/C} - \frac{W_j(t_1,t_2)}{\rho_j/C}\right|, \tag{8}$$

where $W_i(t_1,t_2)$ denotes the service received by flow i during the time interval $(t_1,t_2]$, and $(t_1,t_2]$ is a time interval in which both flows are continuously backlogged.[6]

A scheduler is commonly considered as *fair* if $RFB_{i,j}$ is finite for any couple of flows i and j [8], [14]. However, we showed in Section 2 that a scheduling algorithm can service two *identical* flows at a different rate (thus, behaving unfairly) and still have a finite $RFB_{i,j}$. In fact, according to the ERR analysis reported in [14], it is $RFB_{i,j}^{ERR} = 3L_{\max}\ \forall i,j$, where $L_{\max}$ is the maximum packet length among all the N flows. We then propose a novel and more accurate fairness measure.

Definition 2. *The Generalized Relative Fairness Bound $GRFB_{i,j}$ between any two flows i and j is defined as follows:*

$$GRFB_{i,j} = \max_{t_2>t_1}\left(\frac{W_i(t_1,t_2)}{\rho_i/C} - \frac{W_j(t_1,t_2)}{\rho_j/C}\right) \tag{9}$$

for any time interval $(t_1,t_2]$ in which flow j is continuously backlogged.

We will say that a scheduler is *fair* if it has a finite $GRFB_{i,j}$ for any couple of flows i and j. Intuitively, $GRFB_{i,j}$ measures the maximum lag in the normalized service that a backlogged flow j can get with respect to another one (whether continuously backlogged or not). The key difference between $GRFB_{i,j}$ and $RFB_{i,j}$ is that the first one is the maximum computed on every time interval in which j is continuously backlogged, whereas $RFB_{i,j}$ is computed on every interval in which *both* i and j are backlogged. The following lemma establishes a relationship between $GRFB_{i,j}$ and $RFB_{i,j}$.

Lemma 2. $GRFB_{i,j} \geq RFB_{i,j}\ \forall i,j.$

Proof. Let us prove the inequality by contradiction. Let us assume that there are two flows i and j for which $GRFB_{i,j} < RFB_{i,j}$. This implies that there are t_1^* and t_2^*, $t_2^* > t_1^*$ such that:

$$\left|\frac{W_i(t_1^*,t_2^*)}{\rho_i/C} - \frac{W_j(t_1^*,t_2^*)}{\rho_j/C}\right| > GRFB_{i,j}.$$

We can always choose i and j in such a way that the difference on the left-hand side is positive. Therefore, we have:

$$\frac{W_i(t_1^*,t_2^*)}{\rho_i/C} - \frac{W_j(t_1^*,t_2^*)}{\rho_j/C} > GRFB_{i,j} = \max_{t_2>t_1}\left(\frac{W_i(t_1,t_2)}{\rho_i/C} - \frac{W_j(t_1,t_2)}{\rho_j/C}\right),$$

which clearly cannot hold. □

Lemma 2 allows that $GRFB_{i,j}$ be infinite even if $RFB_{i,j}$ is finite. For instance, $GRFB_{i,j}^{ERR} = \infty$ for any i,j. In fact, in the ERR scenario described in Section 2, assuming as an observation interval $(t_a^2, t_a^{k+2}]$, $k \geq 1$, we would obtain:

$$\frac{W_i(t_a^2,t_a^{k+2})}{\rho_i/C} - \frac{W_j(t_a^2,t_a^{k+2})}{\rho_j/C} = \frac{9}{2}k. \tag{10}$$

Since the right-hand side of (10) cannot be upper bounded, we conclude that $GRFB_{i,j}^{ERR} = \infty$ for any i,j, i.e., ERR is *not* fair according to the $GRFB$ measure. We will now assess the fairness of EBRR by computing its GRFB. Let us first prove some intermediate results.

Lemma 3. *Let us denote with $\underline{C_i^n}$ the value of flow i's credit variable just after flow i transmits its last packet during round n. Let us consider a set of k consecutive rounds—without loss of generality, we number them from 1 to k; let us assume that i is backlogged and eligible at least once in rounds $[1,k]$, and let $\xi(k)$ be the last round during which flow i transmits a packet. Let $W_i(k)$ denote the number of flits transmitted by flow i in the set of k rounds. Then,*

$$W_i(k) \leq k\cdot\phi_i - \underline{C_i^{\xi(k)}}. \tag{11}$$

Proof. We prove the thesis by induction over k. Throughout the proof, we denote with $\overline{C_i^n}$ the value of flow i's credit variable at the *beginning* of round n.

5. Note that, since q is independent of the number of scheduled flows, the complexity of EBRR does not depend on the number of flows as well, regardless of how the *NewRound* function is implemented.

6. Depending on the author, the expression for the RFB may vary a little, usually including/excluding normalization constants. As the RFB is always used for comparison's sake, this is clearly not fundamental.

Base case: $k = 1 \Rightarrow W_i(1) \le \phi_i - \underline{C_i^{\xi(1)}}$.

Since we assumed that i is backlogged and eligible *at least once*, it is $\xi(1) = 1$. In that case, we have $W_i(1) = \overline{C_i^1} - \underline{C_i^1} \le \phi_i - \underline{C_i^1}$ by Lemma 1. This proves the assertion in the base case.

Inductive step: $W_i(k) \le k \cdot \phi_i - \underline{C_i^{\xi(k)}} \Rightarrow W_i(k+1) \le (k+1) \cdot \phi_i - \underline{C_i^{\xi(k+1)}}$.

Suppose that i is not backlogged or not eligible during round $k+1$; in that case, $\xi(k+1) = \xi(k)$, $W_i(k) = W_i(k+1)$ and the thesis obviously holds. We then suppose that i is backlogged and eligible during round $k+1$ and, in that case, $\xi(k+1) = k+1$. Two subcases are given:

1. $\underline{C_i^{\xi(k)}} \ge 0$. According to the EBRR definition, this implies that, at the beginning of round $k+1$, $\overline{C_i^{k+1}} = \phi_i$. We can then write the following expression:

$$\begin{aligned} W_i(k+1) &= W_i(k) + \phi_i - \underline{C_i^{k+1}} \\ &= (k+1) \cdot \phi_i - \underline{C_i^{\xi(k)}} - \underline{C_i^{\xi(k+1)}}, \end{aligned}$$

which proves the inductive step.

2. $\underline{C_i^{\xi(k)}} < 0$. This means that flow i has not been eligible within some rounds from $\xi(k)$ to k. As the flow is eligible at round $k+1$, the following inequality holds:

$$[(k+1) - \xi(k)] \cdot \phi_i > -\underline{C_i^{\xi(k)}}. \tag{12}$$

Since no packet is transmitted from the end of round $\xi(k)$ to the end of round k, it is:

$$W_i(k) = W_i(\xi(k)) = \xi(k) \cdot \phi_i - \underline{C_i^{\xi(k)}}. \tag{13}$$

According to the EBRR definition, either of the two cases is given:

- $\overline{C_i^{k+1}} = \underline{C_i^{\xi(k)}} + [(k+1) - \xi(k)] \cdot \phi_i$. In this case, we can write:

$$W_i(k+1) = W_i(\xi(k)) + \overline{C_i^{k+1}} - \underline{C_i^{k+1}} \le (k+1) \cdot \phi_i - \underline{C_i^{\xi(k+1)}}.$$

- $\overline{C_i^{k+1}} = \phi_i$. This may happen if $\underline{C_i^{\xi(k)}} + [(k+1) - \xi(k)] \cdot \phi_i \ge \phi_i$, in which case:

$$W_i(k+1) = W_i(\xi(k)) + \phi_i - \underline{C_i^{k+1}} \le (k+1) \cdot \phi_i - \underline{C_i^{\xi(k+1)}}.$$

□

Note that, if i is never backlogged or never eligible, it is trivially $W_i(k) = 0$. From Lemma 1 and Lemma 3, the following corollary follows straightforwardly.

Corollary 1. *In k consecutive rounds, a flow can transmit at most:*

$$W_i(k) \le k \cdot \phi_i + (L_i - 1) \tag{14}$$

flits of its traffic.

Corollary 1 states an *upper bound* on the service that a flow may receive in a number of consecutive rounds, whether it is backlogged or not. In order to compute the GRFB for EBRR, we also need a *lower bound* on the service that a backlogged flow may receive in a number of consecutive rounds.

Lemma 4. *Let flow j be always backlogged within $(t_1, t_2]$. Let $R(t)$ denote the ongoing round at time t, and let $R_1 = R(t_1)$ and $R_2 = R(t_2)$; the following inequality holds:*

$$W_j(t_1, t_2) \ge \max[0, (R_2 - R_1 - 1) \cdot \phi_j - (L_j - 1)]. \tag{15}$$

Proof. The worst-case scenario for flow j is the one in which it has the lowest possible credit (i.e., $L_j - 1$, according to Lemma 1) at time t_1, and the maximum possible credit (i.e., ϕ_j, according to Lemma 1) at time t_2. By considering that the credit of a backlogged flow is increased by ϕ_j on each round, the thesis follows straightforwardly. □

We are now ready to prove the main result.

Fairness Theorem. *Let F denote the frame length, L_i and ϕ_i denote the maximum packet length and the quantum for flow i, respectively. Then,*

$$GRFB_{i,j}^{EBRR} \le F \cdot \left(1 + \frac{L_i - 1}{\phi_i} + \frac{L_j - 1}{\phi_j}\right). \tag{16}$$

Proof. Let us consider an interval $(t_1, t_2]$ in which flow j is always backlogged, and let $1 = R(t_1)$ and $k = R(t_2)$. If equality holds into (14) and (15), we have:

$$\begin{aligned} \frac{W_i(t_1,t_2)}{\rho_i/C} - \frac{W_j(t_1,t_2)}{\rho_j/C} &= \frac{k \cdot \phi_i + (L_i - 1)}{\phi_i/F} \\ &- \frac{\max[0, (k-1) \cdot \phi_j - (L_j - 1)]}{\phi_j/F}. \end{aligned} \tag{17}$$

In order to evaluate (17), we treat two cases separately:

Case 1: $\max[0, (k-1) \cdot \phi_j - (L_j - 1)] = 0$. This implies that:

$$k \le \left\lfloor \frac{L_j - 1}{\phi_j} \right\rfloor + 1$$

which, put into (17), yields:

$$\frac{W_i(t_1,t_2)}{\rho_i/C} - \frac{W_j(t_1,t_2)}{\rho_j/C} \le F \cdot \left[1 + \left\lfloor \frac{L_j - 1}{\phi_j} \right\rfloor + \frac{(L_i - 1)}{\phi_i}\right].$$

Case 2: $\max[0, (k-1) \cdot \phi_j - (L_j - 1)] = (k-1) \cdot \phi_j - (L_j - 1)$. In this case, (17) yields:

$$\frac{W_i(t_1,t_2)}{\rho_i/C} - \frac{W_j(t_1,t_2)}{\rho_j/C} = F \cdot \left[1 + \frac{L_i - 1}{\phi_i} + \frac{L_j - 1}{\phi_j}\right].$$

Therefore, in both cases, the right-hand side of is upper bounded by:

$$F \cdot \left(1 + \frac{L_i - 1}{\phi_i} + \frac{L_j - 1}{\phi_j}\right).$$

This completes our proof. □

Note that $GRFB_{i,j}^{EBRR} = RFB_{i,j}^{EBRR}\ \forall\, i, j$.

4.3 Delay Bounds

As previously mentioned in the introduction, in wormhole switching networks, packets can be blocked. In that case, it is not possible to bound the delay of a packet at a node without knowing a bound on the blocking times. The latter, in turn, cannot be easily accounted for, as it depends on several factors, including network topology, routing algorithms, and distribution of the communicating applications on the various nodes. Nevertheless, it is possible to measure the *service delay* of a packet, i.e., the number of flits that are transmitted on the channel before the last flit of the packet is transmitted; the latter clearly does not depend on the blocking time. In the rest of this section, we will compute service delay bounds for EBRR; we will omit the word *service* whenever doing so does not lead to ambiguity. We say that flow i has a *scheduling delay bound* of n flits if the transmission of a packet which is at the head of flow i's queue is completed after at most n flits have been transmitted on the output channel. More formally:

Definition 3. *Given a flow i, its scheduling delay bound is:*

$$D_i = \max_j \{d(p_{i,j}) - h(p_{i,j})\}, \tag{18}$$

where $p_{i,j}$ represents the jth packet from flow i, and $d(p_{i,j}) - h(p_{i,j})$ is the number of flits transmitted on the output link starting from when $p_{i,j}$ reaches the head of flow i's packet queue (either arriving at an empty queue or being promoted after the last flit of the previous packet $p_{i,j-1}$ has been transmitted) up to the time the last flit of $p_{i,j}$ is transmitted.

Note that the delay that a packet experiences at a node is the sum of a *queueing delay* and a *scheduling delay*. Since the latter is only due to scheduling decisions (whereas the former very much depends on the packet arrival or generation process), its upper bound can be used as a figure of merit. As for EBRR's scheduling delay bound, we prove the following theorem.

Scheduling Delay Theorem. *Let t_0 denote the time at which a tagged packet becomes the head packet for flow i. The EBRR scheduling delay is upper bounded by:*

$$D_i^{EBRR} = \sum_{j=1}^{N} L_j + E_i, \tag{19}$$

where E_i is called eligibility delay, and:

$$E_i = \begin{cases} 0 & \text{if flow } i \text{ is eligible at time } t_0 \\ \sum_{j=1}^{N} L_j + \left(\left\lfloor \frac{L_i - 1}{\phi_i} \right\rfloor + 1\right) \cdot & \\ (F - \phi_i) - (L_i + N - 1) & \text{if flow } i \text{ is } \textit{not} \text{ eligible at time } t_0. \end{cases} \tag{20}$$

Proof. If flow i is eligible at time t_0, it is enqueued in the current (i.e., the $|RC|_q$th) active list. Assuming as a worst-case that every other flow is backlogged and eligible at time t_0^-, flow i can be queued after at most $N - 1$ flows and, therefore, the scheduler will service its head packet after at most one packet from every other flow. By considering that each flow (including i) can have a maximum sized packet as its head packet, the thesis follows straightforwardly.

Let us instead assume that flow i is *not* eligible at time t_0. Therefore, it must have been enqueued in the $|EC_i|_q$th active list; by Lemma 1, we know that:

$$EC_i - RC \le 1 + \left\lfloor \frac{L_i - 1}{\phi_i} \right\rfloor. \tag{21}$$

By Corollary 1, during k rounds each flow $j \neq i$ can transmit at most $k \cdot \phi_i + (L_i - 1)$ flits, for an overall length of

$$k \cdot (F - \phi_i) + \sum_{j=1}^{N} (L_j - 1) - (L_i - 1). \tag{22}$$

Therefore, the delay due to eligibility for flow i is upper bounded by:

$$\left(1 + \left\lfloor \frac{L_i - 1}{\phi_i} \right\rfloor\right)(F - \phi_i) + \sum_{j=1}^{N} (L_j - 1) - (L_i - 1). \tag{23}$$

After flow i becomes eligible, it can at most suffer from one packet delay from every other flow. Thus, the upper bound on the scheduling delay for a noneligible flow is again (19). □

In an interconnection network, traffic consists of a mixture of short *control* messages (e.g., cache lines, synchronization messages) and longer *data* messages, which exhibit almost opposite properties. Control messages seldom arrive as part of traffic streams (so that the "flow" which carries them often consists of just one packet) and have a low tolerance for delay—as their timely dispatching is critical for the performance of the end systems. On the other hand, data messages may travel through the network segmented in a (possibly large) stream of packets, and their scheduling delay is expected to be a less critical issue. For this reason, the scheduling delay bound for the *first packet* of a new flow (also referred to as the *start-up latency bound* of the flow [14]) is particularly relevant in interconnection networks. While expressions (19) and (20) apply to all packets in a flow, we can easily derive the start-up latency bound of a flow by considering that in EBRR a flow is eligible as soon as it activates:

$$S_i^{EBRR} = \sum_{j=1}^{N} L_j. \tag{24}$$

We observe that start-up latency of a flow is *independent of the quantum allocation* and, therefore, it is the *same for all flows*.

We compare the start-up latency of EBRR with that of *Weighted ERR* (WERR) [14]. The latter is a variant of ERR, in which each flow is assigned a weight w_i, playing a similar role as EBRR's quantum, and the allowance if computed as $A_i = w_i \cdot (1 + PreviousMaxSC) - SC_i$. In WERR, flows share the channel bandwidth according to their weights. The start-up latency of WERR can be proven to be the following:

$$S_i^{WERR} = L_{\max} \cdot \sum_{\substack{j=1..N \\ j\neq i}} w_j + \sum_{j=1}^{N} L_j - (N-1), \qquad (25)$$

where $L_{\max} = \max_{j=1..N} (L_j)$. Note that this bound is tight, i.e., it is actually possible to build a scenario in which the first packet of a new flow experiences a delay equal to S_i^{WERR}. In order to compare EBRR to WERR, we compute the difference between (25) and (24):

$$S_i^{WERR} - S_i^{EBRR} = L_{\max} \cdot \sum_{j=1..N, j\neq i} w_j - (N-1). \qquad (26)$$

It is straightforward to see that (26) is always positive, reaches a minimum equal to $(L_{\max} - 1) \cdot (N-1)$ if all weights are equal to one (i.e., when WERR reduces to ERR), and grows linearly with the sum of the weights, i.e., with the rate differentiation. Therefore, if the rate requirements differ as little as one order of magnitude, EBRR guarantees a significantly smaller start-up latency bound than WERR. Note that, if EBRR is applied to a *constant-rate server* (i.e., one whose actual forwarding rate is constant), time delay bounds can be easily obtained by the service delay bounds by dividing the latter by the channel speed.

4.4 Trade Off between Performance and Complexity

So far, we have described EBRR assuming that the number of active lists q employed in its implementation is given. We now discuss how the selection of q affects performance and complexity. In EBRR, a flow regains eligibility by increasing its credit at a rate of one quantum per round. A larger q allows the scheduler to keep a record of a larger number of future rounds. Thus, working with a larger q implies allowing quanta to be smaller: in fact, for a given q, we obtain from (7):

$$\forall i = 1..N, \quad \phi_i \geq \frac{L_i}{q-1}. \qquad (27)$$

Working with smaller quanta has two desirable consequences: on the one hand, the frame length (which is the sum of the quanta) is reduced accordingly, thus reducing the GRFB (16). On the other hand, it allows a finer degree of control over rate allocation, as shown by (4).

The above two advantages can be traded off with time-complexity and space occupancy which, as shown in Section 4.1, also depend on q. Depending on the context in which the EBRR algorithm is to be applied, various trade off points can be envisaged. In an interconnection network, where the performance very much depends on the intrarouter latencies and scheduling is implemented in hardware [16], a slightly better fairness and a finer control over bandwidth sharing is unlikely to be worth the price of complicating the scheduling module. Therefore, the trade off point would lean toward a small number of active lists, e.g., few units. On the other hand, in an IP or ATM network, where the intrarouter (intraswitch) delays are often negligible with respect to queueing delays and scheduling is more often implemented in software, it could be feasible to implement EBRR with a higher number of active lists, e.g., in the order of tens or even hundreds, in order to reduce the unfairness and improve control over bandwidth sharing.

TABLE 1
Simulation Parameters

Flow no	1-10	11-20
Packet length (flits)	1-128	1-128
Required rate (Kflit/s)	1	9
ERR weight	1	9
EBRR, SRR quantum (flits)	128	1152

5 Simulations

In this section, we report simulative results that provide a greater insight about the behavior of EBRR and its differences with respect to SRR and WERR. We have implemented the three schedulers in the *ns*-2 simulator [15]. Throughout the simulations, we assume that a flit is 16 bytes long. We have already shown in Sections 2 and 4.2 that WERR (and possibly SRR) can actually exhibit an unfair behavior when flows alternate between idle and backlogged periods. We now show that EBRR provides better fairness than WERR and SRR when flows are constantly backlogged. Moreover, we show that the fairness of EBRR actually improves with the number of active lists employed. We simulate 20 flows, all of which are kept constantly backlogged, sharing a channel with a bandwidth of 12.8 Mbps, i.e., 100 Kflits/s. All flows transmit packets whose length is uniformly distributed between 1 and 128 flits, i.e., $L_i = L = 128$. Flows 1-10 require a rate of 1Kflits per second, whereas packets from flows 11-20 require nine times as much. Weights for ERR and quanta for EBRR and SRR are selected according to Table 1. As for EBRR, we assume that only two active lists are employed, so as to use larger quanta. On the basis of what was outlined in Section 4.4, this implies putting EBRR in the least favorable conditions for a fairness comparison.

Fig. 4 shows the departures of packets from flow 13 in the interval $[14s, 15s]$ in both WERR (Fig. 4a) and EBRR (Fig. 4b). As the figure clearly shows, packets leave in large, evenly spaced bursts under WERR. On the other hand, packet departures are much more irregularly spaced in EBRR, meaning that packets from the same flow do not leave in bursts. SRR behaves the same as WERR and, therefore, we omit plotting the related results. Since fairness is related to the difference in the service received by flows during the same time interval, the burstier the service, the worse the fairness is.

We show this in Fig. 5, which compares the absolute difference in the service received by flows 13 and 14 in the interval $[10s, 90s]$ under ERR and EBRR. In order to improve readability, the absolute difference is averaged on a moving window of 100 packets. The same figure shows the output of a PGPS scheduler, which is a sorted-priority scheduling algorithm whose complexity is $O(N)$.[7] In PGPS, weights are selected as in ERR. As the figure clearly shows, the

7. As explained in Section 2, PGPS might not be suitable for a wormhole switching network, since it requires knowing the length of a packet before taking a decision on its transmission.

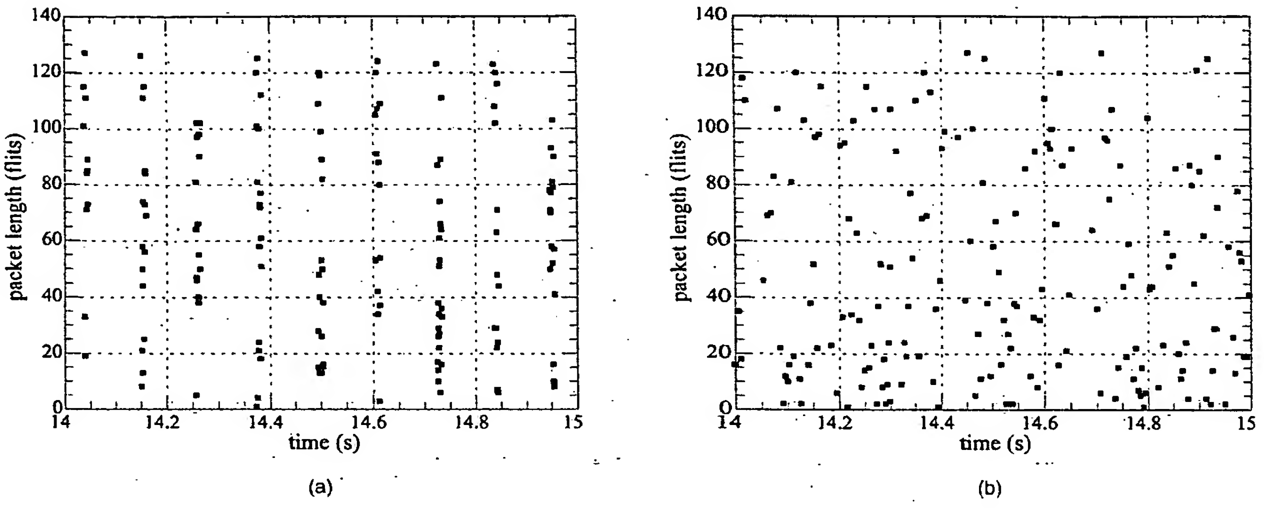

Fig. 4. Service under (a) ERR and (b) EBRR.

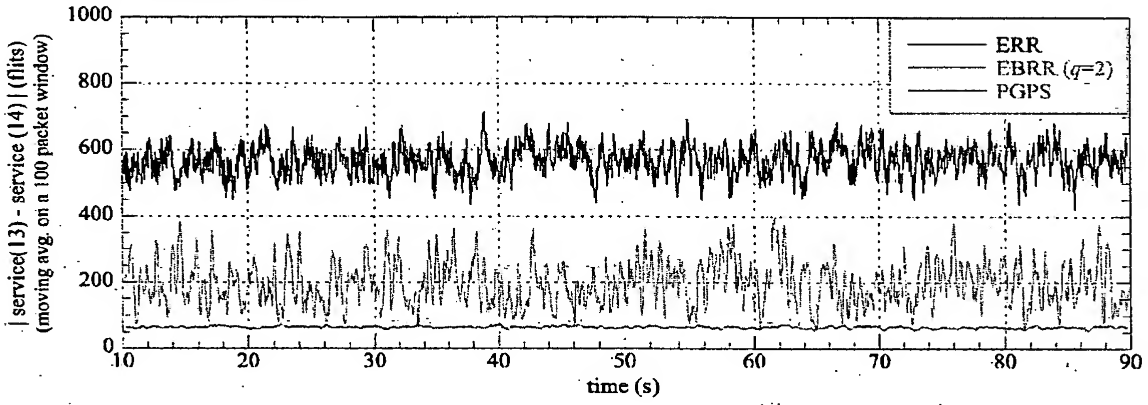

Fig. 5. Absolute difference in the service received by flows 13 and 14 under ERR, EBRR, and PGPS.

difference is considerably smaller in EBRR than in ERR (about 1/3 on average). As would be expected, PGPS fares better than the other two.

In order to show how the number of active lists influences fairness in EBRR, we simulate the latter with the same input flows, with q ranging from 2 to 10. Quanta are allocated as $\phi_i = \lceil L/(q-1) \rceil$ for $1 \leq i \leq 10$, and $\phi_i = 9 \cdot \lceil L/(q-1) \rceil$ for $11 \leq i \leq 20$. We ran each simulation for 2,000 s, and collected the number of bytes that arrived on each flow every 0.2 s. Fig. 6 shows the plots of the average absolute difference in the service received by couples of flows $(3, 4)$ and $(13, 14)$ against q. Confidence intervals are not displayed since they are negligible.

As the figure shows, increasing the number of active lists actually improves fairness. The improvement is more evident for flows with higher rates (i.e., flows 13 and 14 in the figure). In order to demonstrate this further, Fig. 7 shows one more simulative comparison between EBRR and PGPS. This time, we select $q = 17$, which allows EBRR quanta to be reduced by a factor of 16. As the figure shows, in this case the difference is *smaller* in EBRR than in PGPS by an average of 30 percent.

6 Conclusions

In this paper, we have presented a new fair packet scheduling algorithm, called Eligibility-Based Round Robin. Though the latter is explicitly devised to meet the specific

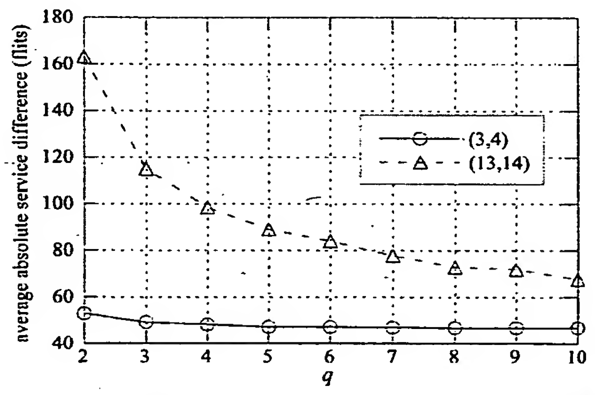

Fig. 6. Average absolute difference in the service received by couple of flows.

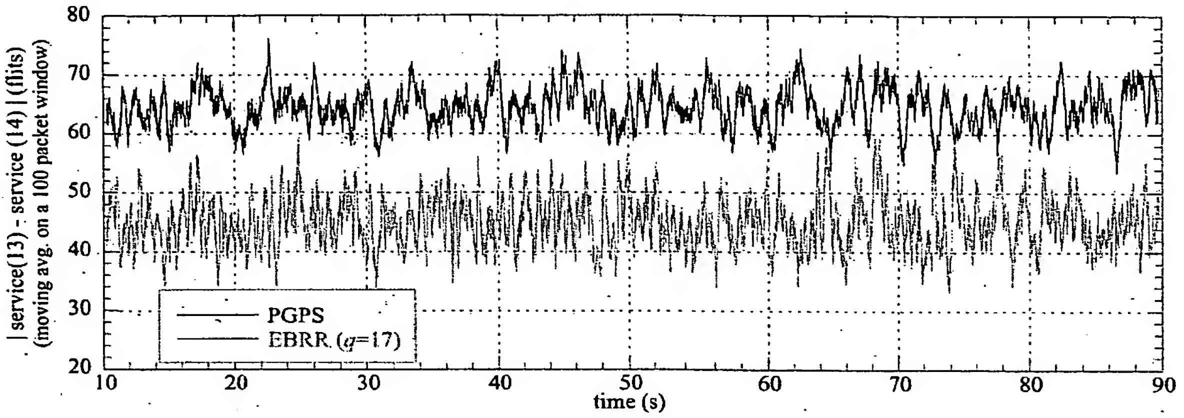

Fig. 7. Absolute difference in the service received by flows 13 and 14 under PGPS and EBRR.

constraints imposed by wormhole switching technology, which is popular in interconnection networks of parallel systems, it can also be employed in packet switching networks such as IP or ATM, as well as to other contexts (e.g., job scheduling in an operating system). We have proven that EBRR has $O(1)$ complexity, and that its scheduling delay bound is tighter than those of existing algorithms of comparable complexity. For the purpose of analyzing the fairness of EBRR, we have investigated the effectiveness of the most commonly used fairness measure, i.e., the Relative Fairness Bound: we have proven that an algorithm can have a finite RFB and still share bandwidth unfairly among flows. We have therefore proposed an alternative fairness measure, called Generalized Relative Fairness Bound, which allows a more precise fairness assessment.

Further work on this topic will include providing a general methodology for computing the GRFB for known classes of scheduling algorithms.

ACKNOWLEDGMENTS

This work has been carried out under the financial support of the Italian *Ministero dell'Istruzione, dell'Università e della Ricerca* (MIUR) under the framework of the project "High Quality Web Systems."

REFERENCES

[1] H. Zhang, "Service Disciplines for Guaranteed Performance Service in Packet Switching Networks," *Proc. IEEE*, vol. 83, no. 10, pp. 1374-1396, Oct. 1995.

[2] D. Stiliadis and A. Varma, "Latency-Rate Servers: A General Model for Analysis of Traffic Scheduling Algorithms," *IEEE/ACM Trans. Networking*, vol. 6, pp. 675-689, Oct. 1998.

[3] D. Stiliadis and A. Varma, "Latency-Rate Servers: A General Model for Analysis of Traffic Scheduling Algorithms," Technical Report CRL-95-38, Univ. of California at Santa Cruz, July 1995.

[4] A.K. Parekh and R.G. Gallager, "A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks: The Single Node Case," *IEEE/ACM Trans. Networking*, vol. 1, pp. 344-357, June 1993.

[5] S.J. Golestani, "A Self-Clocked Fair Queueing Scheme for Broad-band Applications," *Proc. IEEE INFOCOM*, pp. 636-646, June 1994.

[6] P. Goyal, H.M. Vin, and H. Cheng, "Start-Time Fair Queueing: A Scheduling Algorithm for Integrated Services Packet Switching Networks," *IEEE/ACM Trans. Networking*, vol. 5, no. 5, pp. 690-704, Oct. 1997.

[7] J. Bennett and H. Zhang, "Hierarchical Packet Fair Queueing Algorithms," *IEEE/ACM Trans. Networking*, vol. 5, no. 5, pp. 675-689, Oct. 1997.

[8] M. Shreedhar and G. Varghese, "Efficient Fair Queueing Using Deficit Round Robin," *IEEE/ACM Trans. Networking*, vol. 4, pp. 375-385, June 1996.

[9] G. Parulkar, H. Adiseshu, and G. Varghese, "A Reliable and Scalable Striping Protocol," *Proc. ACM SIGCOMM*, pp. 131-141, Aug. 1996.

[10] L. Lenzini, E. Mingozzi, and G. Stea, "Aliquem: A Novel DRR Implementation to Achieve Better Latency and Fairness at $O(1)$ Complexity," *Proc. IEEE 10th Int'l Workshop Quality of Service*, pp. 77-86, May 2002.

[11] L. Lenzini, E. Mingozzi, and G. Stea, "A Unifying Service Discipline for Providing Rate-Based Guaranteed and Fair Queueing Services Based on the Timed Token Protocol," *IEEE Trans. Computers*, vol. 51, no. 9, Sept. 2002.

[12] L. Lenzini, E. Mingozzi, and G. Stea, "Packet Timed Token Service Discipline: A Scheduling Algorithm Based on the Dual-Class Paradigm for Providing QoS in Integrated Services Networks," *Computer Networks*, vol. 39, no. 4, pp. 363-384, July 2002.

[13] S.S. Kanhere, A.B. Parekh, and H. Sethu, "Fair and Efficient Packet Scheduling in Wormhole Networks," *Proc. Int'l Parallel and Distributed Processing Symp.*, May 2000.

[14] S.S. Kanhere, H. Sethu, and A.B. Parekh, "Fair and Efficient Packet Scheduling Using Elastic Round-Robin," *IEEE Trans. Parallel and Distributed Systems*, vol. 13, no. 3, Mar. 2002.

[15] The Network Simulator—ns-2, http://www.isi.edu/nsnam/ns/, 2003.

[16] J. Duato, S. Yalamanchili, and L. Ni, *Interconnection Networks: An Engineering Approach*. IEEE Computer Soc. Press, 1997.

[17] N.J. Boden, D. Cohen, R.E. Felderman, A.E. Kulawik, C.L. Seitz, J.N. Seizovic, and W.K. Su, "Myrinet—A Gigabit-per-Second Local-Area Network," *IEEE Micro*, pp. 29-36, Feb. 1995.

[18] J. Rexford and K.G. Shin, "Support for Multiple Classes of Traffic in Multicomputer Routers," *Proc. Workshop Parallel Computer Routing and Comm.*, pp. 116-130, May 1994.

[19] H. Sethu, C.B. Stunkel, and R.F. Stucke, "IBM RS/6000 SP Large System Interconnection Network Topologies," *Proc. Int'l Conf. Parallel Processing*, Aug. 1998.

[20] S. Senapathi, D.K. Panda, D. Stredney, and H.-W. Shen, "A QoS Framework for Clusters to Support Applications with Resource Adaptivity and Predictable Performance," *Proc. IEEE 10th Int'l Workshop Quality of Service*, pp. 180-190, May 2002.

[21] S.-C. Tsao and Y.-D. Lin, "Pre-Order Deficit Round Robin: A New Scheduling Algorithm for Packet Switched Networks," *Computer Networks*, vol. 35, pp. 287-305, Feb. 2001.

Luciano Lenzini received a degree in physics from the University of Pisa, Italy. He joined CNUCE (Centro Nazionale Universitario di Calcolo Elettronico), an Institute of the Italian National Research Council (CNR), in 1970. In 1994, he joined the Department of Information Engineering at the University of Pisa as a full professor. His current research interests include the design and performance evaluation of MAC protocols for wireless networks and the quality of service provision in integrated and differentiated services networks. He is a member of the IEEE.

Enzo Mingozzi received the PhD degree in computer systems engineering in 1995 and 2000, respectively, from the University of Pisa. He is an assistant professor in the Department of Information Engineering at the University of Pisa. Since 1995, he has been involved in several national and international projects including, among others, Eurescom P810, "Wireless ATM access and advanced software techniques for mobile networks architecture," and IST WINE GLASS, "Wireless IP NEtwork as a Generic platform for Location Aware Service Support." He also took part in the standardization process of the HIPERLAN/2 and HIPERACCESS protocols, in the framework of the ETSI project BRAN. He served as the tutorial chair on the European Wireless 2002 conference committee. His current research interests focus on the design and analysis of MAC protocols for wireless networks, and QoS provisioning and service integration in computer networks.

Giovanni Stea received the Laurea and the PhD degrees in computer systems engineering from the University of Pisa, Italy, in 1999 and 2003, respectively. Since December 2002, he has been a research fellow in the Department of Information Engineering at the same university. He has been and is involved in national and european research projects. He has served as a member of the technical program committee and local arrangements chairman for the European Wireless 2002 International Conference. His current research interests include network scheduling, quality of service in multiservice networks, and network architectures.